Towards a broader view on software architecture analysis of flexibility

Lassing, N.H.; Rijsenbrij, D.B.B.; van Vliet, H.

published in
Proceedings of the Asian-Pacific Software Engineering Conference (APSEC'99) Takamatsu, Japan, 1999
1999

document version
Publisher's PDF, also known as Version of record

Towards a Broader View on Software Architecture Analysis of Flexibility

Nico Lassing, Daan Rijsenbrij and Hans van Vliet
Faculty of Sciences, Vrije Universiteit, Amsterdam
{nlassing, daan, hans}@cs.vu.nl

Abstract

Software architecture analysis helps us assess the quality of a software system at an early stage. In this paper we describe a case study of software architecture analysis that we have performed to assess the flexibility of a large administrative system. Our analysis was based on scenarios, representing possible changes to the requirements of the system and its environment. Assessing the effect of these scenarios provides insight into the flexibility of the system. One of the problems is to express the effect of a scenario in such a way that it provides insight into the complexity of the necessary changes. Part of our research is directed at developing an instrument for doing just that. This instrument is applied in the analysis described in this paper.

1. Introduction

Recently, there has been a wide interest in the study of software architecture. A system's software architecture captures early design decisions, which have a major impact on the quality of the resulting system. It is very hard, if not impossible, to change these decisions later on. Therefore, it is essential that we judge the appropriateness of these decisions at an early stage. Software architecture analysis enables us to do so.

Currently, ideas about analyzing software architectures are beginning to evolve (see [5] and [1]). A number of authors have reported on methods for software architecture analysis of flexibility, such as [7] and [3]. Both methods use scenarios to capture possible events in the life of a system and evaluate the flexibility of the system by evaluating the effect of these scenarios. Their main difference is the way in which the effect of scenarios is evaluated and expressed. SAAM evaluates the effect of a scenario by investigating which architectural elements are affected by that scenario. Bengtsson and Bosch also predict the effort needed to implement the scenario by estimating the size of these components and the extent to which they are affected.

In the case study presented here, we used scenarios to analyze the flexibility of MISOC2000, a large administrative system developed by the Dept of Defense Telematics Agency (in Dutch: Defensie Telematica Organisatie or DTO) for the Dutch Dept of Defense (DoD). The purpose of this case study was to gain insight into the factors that influence the complexity of changes for this class of information systems. We found that the number of components affected and their respective size are not the most important factors that influence the complexity of changes for administrative systems; other factors are more important. Based on our experiences, we have defined a measurement instrument that includes these factors.

The definition of software architecture we use in this paper is based on the one given in [2]. They define the software architecture of a program or computer system as the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them. We have decided to extend this definition because it only focuses on the internals of a system. We have found that for architectural analysis the external environment is just as important. In our view, the description of the software architecture should consist of two parts. One part should focus on the environment of the system, which we will call the 'macro architecture'. The other part should cover the internal structure of the system, and will be called the 'micro architecture'.

The remainder of this paper is divided into three sections. Section 2 introduces MISOC2000 and describes its software architecture, section 3 contains the analysis of the flexibility of MISOC2000 and our conclusions are given in section 4.

2. Case study: MISOC2000[1]

Our case study concerns the software architecture analysis of a system called MISOC2000, which is currently being developed by DTO. MISOC2000 will be used by fifteen training centers of the various services of the Dutch army for the registration of data concerning their courses and students. These training centers are

[1] MISOC is short for Management Information System for Training Centers (in Dutch: Management InformatieSysteem voor de OpleidingsCentra)

separate organizational units, responsible for their own operating results. They are located throughout the Netherlands and part of Germany and each of them belongs to exactly one branch of military service. The MISOC2000 project is funded by their coordinating department and DTO is the main contractor.

This section describes the software architecture of the MISOC2000 system. It is divided into two parts. The first part, presented in section 2.1, covers the macro architecture of MISOC2000, i.e. the position of MISOC2000 in its environment. The second part, presented in section 2.2, covers MISOC2000's micro architecture, i.e. its internal structure.

2.1. The macro architecture of MISOC2000

MISOC2000 will not be an isolated system, because it has to be integrated with other systems that are already used by the training centers. The macro architecture describes these systems and their relationships to MISOC2000. By making a distinction between systems that are owned by the training centers and those that are owned by others, we can distinguish changes that can be made autonomously by the training centers from changes for which coordination with other organizational units is necessary.

We will start our description of the macro architecture by focusing on the systems of a single training center. This description applies to all training centers, because all of them use the same set of systems. Each training center has a number of systems with which MISOC2000 has to be integrated. The relationships between MISOC2000 and other systems can take various forms. Similar to [6] we have identified three types of relationships, which are in order of increasing integration: (1) data exchange through file transfer, (2) access to persistent data and (3) call relationship. However, more integration between systems leads to stronger dependencies between systems and stronger dependencies between systems make it harder to change one of these systems. In the first situation, the dependency between systems is limited to the structure of the files they exchange. In the second situation, the dependency between systems consists of the structure of the persistent storage. In the third situation, the dependency between the systems is extended to the application logic. So, the degree of dependency between systems determines their mutual flexibility.

In Figure 1, the systems of a single training center are shown, as well as the type of relationship they have to MISOC2000. We will now briefly describe the various systems mentioned in this figure. The course development system (GOOS) is used for developing new courses. To do so, it uses information from MISOC2000, such as the number of registrations for a course and the availability of locations. After new courses have been developed with

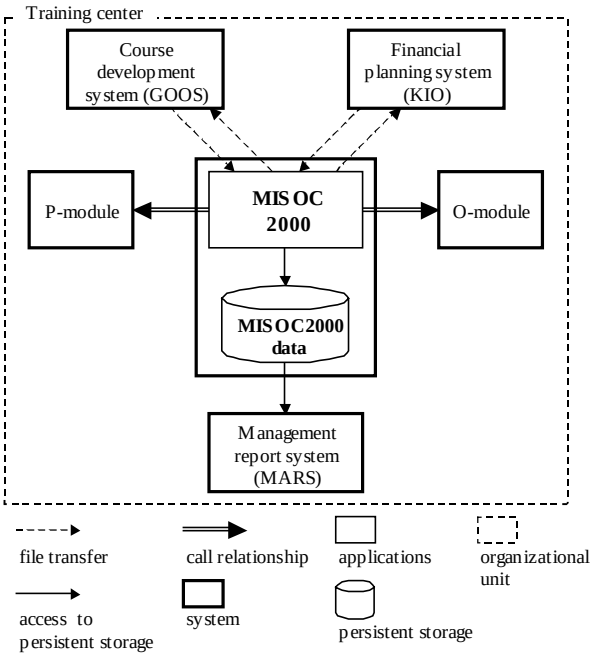

Figure 1. Systems of a training center

GOOS, they are imported into MISOC2000. From then on it is possible to register students for these courses. The data exchanges between MISOC2000 and GOOS consist of files being imported and exported. This means that MISOC2000 and GOOS can be adapted independent of each other, as long as the structure of the files they exchange is unaffected.

The next system is the financial planning system (KIO), which is used for calculation of the costs. KIO feeds MISOC2000 with information concerning cost centers and retrieves information from MISOC2000 concerning the organization, instructors, locations, and resources. These data exchanges take the form of file transfers. So, like GOOS, KIO is rather independent of MISOC2000.

The management reporting system (MARS) is a management information system that is used for generating various management reports. This system is implemented using a COTS report tool. This tool directly accesses the MISOC2000 database to retrieve information. As a result, MARS is independent of the implementation of MISOC2000 and it will be unaffected by changes to MISOC2000 that do not affect its database.

The systems we have mentioned so far are all owned and maintained by the training centers, or their coordinating department. The two remaining systems in Figure 1, the P-module and the O-module, are owned and maintained by a central department. In the evaluation in section 3 we will assess how this notion of ownership affects the flexibility of these systems.

The P-module is a part of the human resource (HR) information system. The HR system stores information

about employees, such as name, rank and qualifications. This information is maintained both at the central level for the whole DoD, and at the local level for each unit. At the local level, each unit uses an instance of the P-module for managing the information of the employees of that unit. Periodically, the central system feeds the P-module of each unit with information concerning the employees of that unit using file transfer. These downloads are one-way only, so global updates to the human resource information are only possible at the central HR system. However, the P-module does provide facilities for performing updates, but these changes are not carried through to other units. This enables units to register temporary staff.

In fact, the P-module plays two different roles, namely as a stand-alone system for managing personnel information and as a 'service' for other systems to access personnel information. MISOC2000 uses the P-module in the latter role, mostly to retrieve information about instructors. When MISOC2000 invokes the P-module, one of the applications of the P-module is started on the user's workstation and control is transferred to that application. After the user has performed the necessary actions, the application is closed and control is returned to MISOC2000. Other systems use the P-module in similar ways. The main drawback of this approach is that it results in strong dependencies between the P-module and these other systems. In section 3 we will touch on the consequences of these dependencies.

We will be brief on the O-module and its central part, because their structure is similar to the P-module and the central HR system. The function of the O-module is to provide access to information concerning the organization and its resources. Just like the human resource information, this information is stored at both the central and the local level and the central mainframe performs periodical downloads to local instances of the O-module.

So far, we have discussed the relationships of MISOC2000 with systems within a training center. However, MISOC2000 is also related to one system outside the training centers, namely PICO (Planning and Development System for Courses and Training). PICO gathers the course information of all training centers and enables their customers, i.e. all organizational units, to enroll employees for these courses. Like MISOC2000, PICO is owned by the coordinating organization of the training centers. The structure of PICO and its relationship with MISOC2000 is shown in Figure 2.

Several flows of information can be distinguished in this figure, all of which are file transfers. The information concerning the courses is transferred from the training centers to a central server, the PICO server. The customers of the training centers use a local system, 'PICO customer', to retrieve this information and enroll their employees for these courses. Finally, these

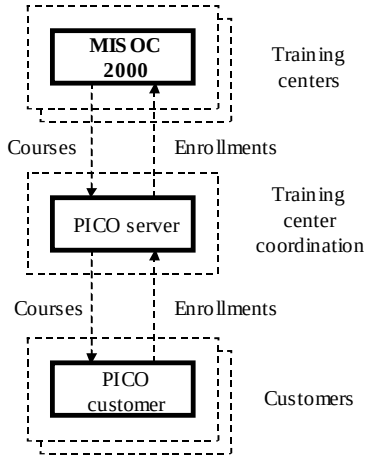

Figure 2. MISOC2000 and PICO

enrollments are transferred from the PICO server to MISOC2000 at the appropriate training center.

A number of the systems we have mentioned so far are used at different locations. To make sure that these systems operate correctly in the technical environment at each location, the DoD has defined the LAN2000 standard. This standard sets the configuration of both client and server machines, e.g. the hardware, the operating system and the database management system. Creating a uniform technical environment removes the need to develop multiple versions of a system to run at different locations, simplifying configuration management. In the analysis in section 3 we touch on the drawbacks of this type of standardization.

2.2. The micro architecture of MISOC2000

MISOC2000 is created with COOL:Gen, an enterprise CASE tool developed by Sterling Software. This tool uses models and code diagrams to specify the behavior of a system, independent of its target technical environment. Based on these models and code diagrams, COOL:Gen can generate the source code and the database schemes of a system for a number of technical environments (compiler, operating system, transaction-processing monitor and database management system). After that, this source code is compiled to create executables for the target environment. Finally, these executables are installed in their target environment, along with a set of run-time files specific for that environment. These run-time files are used by all COOL:Gen generated systems for things like communication and screen-handling.

In COOL:Gen the whole system is stored in one model, but this model consists of seven submodels. The choice of subsystems is driven by the processes of the training centers: each subsystem supports a specific group of users. The following subsystems are recognized:

1. **Product**: formulating course catalogs and production

plans for a training center
2. **Sales**: distribution of course catalogs and recording agreements with customers
3. **Student**: registration of student information
4. **Programming**: creating short-term schedules
5. **Logistics**: management of the availability of locations and items
6. **Economics**: exporting cost information to KIO and importing information about cost centers from KIO
7. **Personnel**: an extension of the P-module to record personnel information specific for training centers

These subsystems communicate through a shared database. Figure 3 shows the subsystems, the information they share and their communication with the systems in the environment.

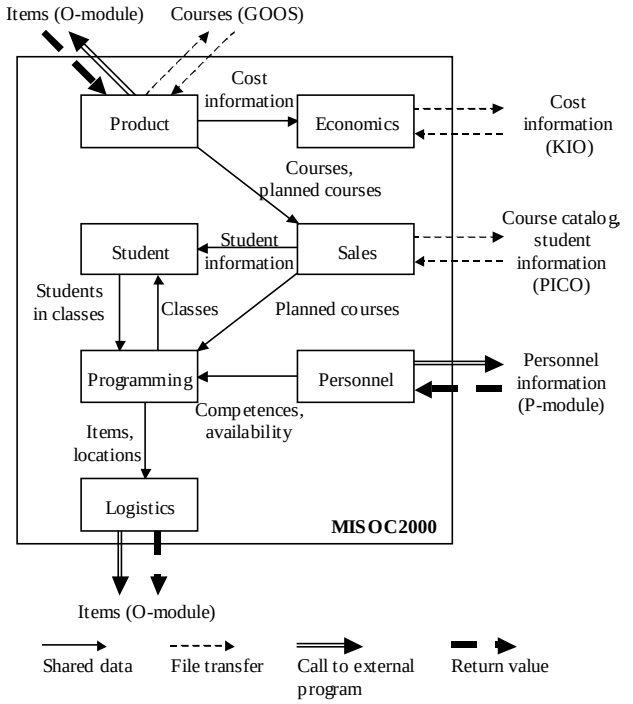

Figure 3. The subsystems of MISOC2000

There is also an eighth subsystem, called 'General'. Although its name suggests otherwise, this subsystem is not aimed at supporting a specific group of users. Instead, it is used for administrative purposes, like maintenance of authorization and configuration data. Information recorded by this subsystem is used by all other subsystems. It was omitted to enhance readability.

Orthogonal to this division in subsystems, MISOC2000 is also divided into three layers. The first layer consists of a number of client executables, which are installed on the users' workstations. The second layer consists of a number of server executables, which are installed on an application server. The third layer consists of the database tables that are placed on the database management server. This layering spreads the required processing over a number of machines.

A similar approach is used for other DoD systems that were created with COOL:Gen, such as the P-module and the O-module. Many people within the DoD use several of these systems. As a result, many users' workstations contain the client executables of a number of systems, which have to share the set of COOL:Gen run-time files.

MISOC2000 is protected from unauthorized use by an authorization mechanism. The authorization strategy that is employed is function-oriented, i.e. groups of users are authorized to perform certain sets of functions. The authorization mechanism consists of a number of elements. The first element is the maintenance of the authorization data. As mentioned before, this function is performed by the subsystem 'General'. The second element is the storage of authorization data. This function is performed by the MISOC2000 database server, which has a separate database for authorization data. The next element is the authentication client that logs users in to and out of MISOC2000. It consists of a small application created with COOL:Gen that is installed on each user's workstation, which registers a user with the database. This authentication client is also used for other systems created with COOL:Gen. The final element of the authorization mechanism is the authorization of functions. To do so, each function checks the authorization database to see whether the current user is authorized to perform that function. Figure 4 shows the relationships between the various elements.

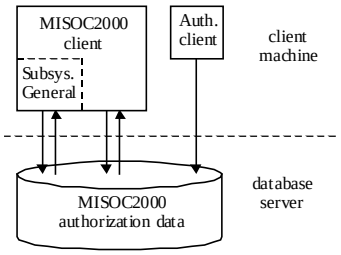

Figure 4. The elements of authorization

Although COOL:Gen is aimed to provide platform independence, it does support the use of OCX-controls[2], which are only usable in a limited number of technical environments. In MISOC2000, the subsystem 'Programming' contains such an OCX-control for showing a timetable. The decision to use this component was driven by the fact that it was available from an external supplier and using it saves a lot of time during development. As a consequence, the advantages of COOL:Gen with respect to portability are not fully exploited. An additional drawback is that the component is owned by an external supplier, which means that the DoD is dependent on this supplier for this component.

[2] An OCX-control is software component that is specific for the Microsoft Windows environment.

3. Analysis of flexibility

In our analysis we focus on the flexibility of MISOC2000. We define flexibility as the ease with which systems can be adapted to changes. These changes are not limited to internal aspects of a system. We found that the environment is an important source of changes as well.

The method we use for our analysis is based on the Software Architecture Analysis Method or SAAM ([7]). This method consists of three major steps:

1. Describe the software architecture in sufficient detail
2. Develop relevant scenarios
3. Evaluate the effect of scenarios

Although the steps are listed here as though they are performed sequentially, they are not. The first two steps, for instance, have to be performed in parallel, for two reasons. First, the description of the software architecture should cover the aspects mentioned in the scenarios. Second, it is very hard to define scenarios when you are not sufficiently familiar with the system and its software architecture. So, the steps are not necessarily performed in the above-mentioned order. Nevertheless, to enhance the comprehensibility of our analysis we present them as discrete, sequential steps.

The first step has already been discussed in detail in section 2. Section 3.1 lists the scenarios we identified and describes their effect on the system. In section 3.2 we introduce the measurement instrument we have developed for expressing the effect of scenarios and apply this instrument to the scenarios of section 3.1. Section 3.3 contains an evaluation of the analysis.

3.1. Scenarios and their effect

The central steps in the analysis of software architectures for flexibility are capturing potential changes in scenarios and evaluating their effect. The scenarios make flexibility tangible and evaluating their impact demonstrates how well they are supported by the software architecture. It is essential to find those changes that are likely to happen in the life of the system. The scenarios used in this analysis were established through interviews we had with various stakeholders of the system. These interviews revealed that adaptations to the system are not only brought about by changes in the requirements, but also by changes in its environment. So, our list of scenarios contains both types of changes. For each scenario we have indicated its most likely initiator.

The next step was to assess the effect of the scenarios. To this end, we interviewed members of the MISOC2000 development team and stakeholders of some of the other systems. The results are described below.

Scenario 1: What happens when a branch of military service replaces Windows NT 4.0 by Windows 2000?

This situation could occur every time a new version of an operating system is released. The situation that one individual service changes its operating system is in fact highly undesirable, because it would require that a number of systems, including MISOC2000, be regenerated and recompiled for this service only. This leads to different versions of the same system, which increases the complexity of configuration management and jeopardizes the interoperability between services. The LAN2000 standard is aimed at avoiding just that. An organizational entity should only change its operating system when the LAN2000 standard is changed. These decisions are made for the entire DoD. As a result, the individual organizational entities have limited control over these decisions and once they have been made they have to follow. So, in this situation flexibility is partly sacrificed for reduced complexity and increased interoperability.

Scenario 2: What happens when the DoD changes the operating system in LAN2000 from Windows NT 4.0 to Unix (for both workstations and servers)?

For MISOC2000 this means that it has to be regenerated and compiled for this new platform. On the server side, this should not be a very large problem, because the server applications of MISOC2000 do not use any platform-specific features. The MISOC2000-applications on the client side, however, do use platform-specific features. The subsystem ‘Programming’ uses an OCX-control, which is not usable in a UNIX-environment. This means that either the external supplier has to supply a similar component for this platform or that such a component has to be created. Although this probably requires a lot of work, it is the only component of MISOC2000 that is affected.

However, MISOC2000 does not exist in isolation. The other systems in its environment have to be ported to the new platform as well. For some of these systems this may prove very hard, because they have to be reimplemented. In addition to the effort that is needed to adapt the individual systems, effort is also needed for coordinating the various changes. This was already recognized by Brooks back in the 1970s (see [4]). He claims that developing and maintaining a system that is related to other systems, costs three times as much as developing and maintaining an isolated system. Although the factor three may not be entirely correct, developing and adapting integrated systems is inherently more complex. So, even though portability seems to be taken care of for MISOC2000, the dependencies with other systems make that it is very hard to change the technical environment.

Scenario 3: What happens when a new version of COOL:Gen is used for MISOC2000?

In section 2.2, we mentioned that each system

developed with COOL:Gen needs a set of run-time files on every machine that contains executables of that system. These run-time files are specific for a version of COOL:Gen. So, when a new version of COOL:Gen is used, these run-time files have to be upgraded as well. However, if the run-time files are upgraded on the workstations of the users of the training centers, the other COOL:Gen created systems on these workstations, the authorization client, the P-module and the O-module, have to be migrated to this new version as well. Otherwise, version conflicts arise. But if these systems were only upgraded at the training centers, they would exist in two versions: one for the training centers and one for the rest of the DoD. We saw earlier that this is regarded undesirable. Therefore, the authorization clients, the P-module and the O-module of every unit of the DoD have to be migrated to this new version of COOL:Gen, including their run-time files. This means that all systems created with COOL:Gen that share a machine with the authorization client, the P-module or the O-module have to be upgraded as well. Eventually, every system that was created with COOL:Gen has to be upgraded. So, when MISOC2000 uses a new version of COOL:Gen, this implies that every system created with COOL:Gen should be regenerated, recompiled, tested and deployed.

Scenario 4: What happens when the authorization client is changed?

The authorization client is an independent application created with COOL:Gen that is used to log users in to and out of MISOC2000. When a user logs in to MISOC2000, the authorization client registers this in the authorization database. No direct communication takes place between MISOC2000 and the authorization client: MISOC2000 just queries the database to see which user is logged in. As a result, MISOC2000 is unaffected by changes to the authorization client that do not affect its database.

Scenario 5: What happens when the user interface style of the P-module is changed?

As mentioned in section 2.1, the user is confronted with the user interface of the P-module when MISOC2000 needs information from the P-module. So, a change in the style of interaction of the P-module causes inconsistencies in the interaction style of MISOC2000. This matter could be resolved by adapting MISOC2000 to this new style.

This situation actually occurred during the development of MISOC2000. It appeared to be very difficult to adapt the style of all its user interface elements. To explore these difficulties, it is necessary to explain how DTO handles user interface styles. DTO propagates the use of a uniform interface style for all systems, by making available a COOL:Gen template that incorporates this style. Initially, MISOC2000 was also based on this template. The problem that arose was that, once a COOL:Gen project is created, its initial template cannot be changed. This meant that in order to adapt the user interface style of MISOC2000 each of its user interface elements had to be adapted by hand. This was considered not worth the extra effort, so now there is a small variation in the user interface style of the P-module and MISOC2000.

Scenario 6: What happens when the external supplier changes the interface style of the timetable component?

This has no impact on MISOC2000 whatsoever, because it is not compulsory to use the new version of the component in MISOC2000. This is the main difference between this timetable component and the P-module in the previous scenario. MISOC2000 is always confronted with the latest version of the P-module.

Scenario 7: What happens when PICO is used for transferring course results to the P-module of the organizational unit of a student?

At present, the course results of a student are transferred to his or her organizational unit by hand, where they are entered into the P-module. Because PICO is already used for passing enrollments from a unit to a training center, it could also be used for automatically transferring the results back to the P-module of this unit. In fact, PICO customer and PICO server are already prepared to handle these transfers. Only MISOC2000 has to be adapted so that it can automatically export these results to PICO. In MISOC2000, the link to PICO is centralized in the subsystem ‘Student’ that also maintains the information concerning results. So, this is the only subsystem that has to be adapted.

Scenario 8: What happens when the processes of the training centers are changed?

We mentioned in section 2.2 that the processes of the training centers drove the division of MISOC2000 in subsystems. This division was chosen in such a way that most tasks could be performed using a single subsystem. To preserve this concept after the processes change, it is necessary to modify the division in subsystems. So, this scenario causes changes to the micro architecture.

Scenario 9: What happens when a number of services have to cooperate in one training center?

At present, a training center always belongs to just one service. This scenario does not change this situation. The only thing that changes is that instructors and assets of one service are allocated to a training center of another service. This means that they have to be entered in the P-module or O-module of this training center as local data. So, MISOC2000 is unaffected by this scenario.

Table 1. Results of the scenarios

		Macro architecture level			Micro architecture level		
	Initiator of scenario	Impact level[3]	Multiple owners	Version conflict[4]	Impact level[3]	Multiple owners	Version conflict[4]
cenario 1	A service	3	+	3	1	-	3
cenario 2	DoD	3	+	4	2	+	1
cenario 3	Training centers	3	+	4	1	-	1
cenario 4	DoD	2	-	1	1	-	1
cenario 5	Central HR dept.	2	-	2	1	-	2
cenario 6	External supplier	1	-	1	1	+	1
cenario 7	Training centers	1	-	1	2	-	1
cenario 8	Training centers	1	-	1	4	-	1
cenario 9	Training centers	1	-	1	1	-	1
cenario 10	Training centers	1	-	1	1	-	1

Scenario 10: What happens when training centers have to share their assets (locations, vehicles, etc.)?

This scenario is similar to the previous one, except that in this current scenario the training centers lose part of their autonomy. To implement this scenario would require that the instances of MISOC2000 at the various training centers be connected. This would have an enormous impact on the macro architecture of MISOC2000. Alternatively, the matter could be solved outside the system, by agreements between training centers about the use of assets. The DoD has a strong preference for the latter solution.

3.2. A measurement instrument for scenarios

One of the main problems in the software architecture analysis of flexibility is to express the effect of a scenario in a systematic way. SAAM is not very clear at this point. Therefore, we have developed a measurement instrument for doing so, which includes a number of measures that determine the complexity of changes required for a scenario. These measures were identified in consultation with developers.

The first measure affecting the complexity of a scenario is its impact, i.e. the magnitude of the required adaptations. In [8] we used the following four levels to express the impact of a scenario on a system:

1. Scenario has no impact
2. Scenario affects one component
3. Scenario affects several components
4. Scenario affects the software architecture

To be able to draw a distinction between the effect of a scenario on a system and the effect on its environment, we will make a distinction between the impact of a scenario at the macro architecture level and the impact at the micro architecture level. At the macro architecture level the components of level 2 and 3 represent systems and at the micro architecture level they represent components or subsystems. The impact of a scenario on the system itself, MISOC2000 in this case, is expressed only at the micro architecture level, not at the macro architecture level.

The complexity of a scenario is also influenced by the notion of ownership, because a scenario is more complex when multiple stakeholders are involved. Not only because of the additional coordination that is required between these parties, but also because all stakeholders have to be persuaded to implement the necessary changes. Ultimately, this could mean that a scenario is not feasible.

An additional factor influencing the complexity of changes is whether a scenario leads to the presence of different versions of some architectural element. Different versions of an architectural element may introduce a number of difficulties. Eventually, this may result in changes to architectural elements that were initially unaffected by a scenario. We have distinguished four levels of difficulties related to versions:

1. No problems with different versions
2. The presence of different versions is undesirable, but not prohibitive
3. The presence of different versions creates difficulties related to configuration management
4. The presence of different versions creates conflicts

So, our instrument includes three measures to express the effect of a scenario. The first measure provides insight into the required changes, the second measure indicates whether coordination between stakeholders is required and the third measure will help us identify any unintentional side effects of scenarios. In Table 1 we use this instrument to rate the effect of scenarios we found.

Table 1 leads us to the following observations. Scenarios 1 and 2 are initiated outside the training centers, but affect the micro architecture of MISOC2000. This means that MISOC2000 has to follow these scenarios,

[3] 1 = no impact, 2 = one component affected, 3 = several components affected, 4 = architecture affected

[4] 1 = no version problems, 2 = presence of multiple versions is undesirable, 3 = presence of multiple versions complicates configuration management, 4 = presence of multiple versions creates conflicts

although they may not be immediately beneficial to the training centers. Scenario 3 represents the reverse situation: it is initiated by a training center but affects architectural elements of other owners as well. As a result, this scenario can only be performed in consultation with others. Scenario 4 is an uncomplicated scenario that affects just one system. Scenario 5 also affects just one system, but it introduces a version conflict at the same time. This conflict is not so serious that other systems have to be adapted as well. As a result, some inconsistencies will remain. We can be short on scenarios 6, 9 and 10, because they do not affect MISOC2000 at all. Scenarios 7 and 8, on the other hand, do affect MISOC2000, but their impact is limited to the micro architecture level. This does not mean that they are easier to perform, but in any case they can be performed autonomously by the training centers.

3.3. Evaluation of the analysis

The principal shortcoming of our approach is that the factors included in the measurement instrument are not entirely comparable. Another shortcoming, which is common for all scenario-based methods, is that you often do not know whether the scenarios found really represent those changes that are likely to happen in the life of a system. Consequently, the results should be interpreted with care. We feel that the instrument is most useful as an aid in the analysis of flexibility.

4. Conclusion

In this paper we have presented a case study of software architecture analysis. The purpose of this case study was to explore the possibilities and difficulties of architecture analysis of flexibility for administrative systems. To this end, we have used an existing technique, SAAM, to analyze the flexibility of MISOC2000, a large administrative system that is built for the Dutch Dept of Defense. This has taught us a number of things. Firstly, we have found that the environment plays an important role in the analysis of flexibility. The environment is not only a source for changes, but it can also complicate the implementation of changes. Therefore, we found it useful to view the software architecture of a system at two levels: the internal structure of the system (the 'micro architecture') and the role of the system in its environment (the 'macro architecture'). Secondly, we have found that ownership is a matter of concern for flexibility. Scenarios that affect architectural elements of different owners are more complicated to perform than those that affect architectural elements of a single owner. Thirdly, we have experienced that the presence of multiple versions may extend the impact of changes.

Based on our findings, we have defined a measurement instrument to express the effect of scenarios. This instrument draws a distinction between the effect of a scenario on the micro architecture and its effect on the macro architecture. For both the micro architecture and the macro architecture, the instrument indicates the impact of a scenario, whether multiple owners are involved and whether it leads to version conflicts. Applying this instrument helps us gain insight into the complexity of scenarios. The principal shortcoming of the instrument is that it includes a number of measures that are not fully comparable. In further research we will use this instrument again to see whether all aspects relevant to the complexity of changes are included.

Acknowledgements

This research is mainly financed by Cap Gemini Netherlands. We thank DTO and the Dutch Dept of Defense for their cooperation. We are especially grateful to Reinoud Sicking, Gabby Niemantsverdriet and Peter Braat of DTO and Hans van Grinsven of the Dutch Army for their time and their comments.

References

[1] G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northrop, A. Zaremski *Recommended Best Industrial Practice for Software Architecture Evaluation*. 1997. CMU/SEI-96-TR-025.

[2] L. Bass, P. Clement and R. Kazman . *Software Architecture in Practice*. 1998. Addison Wesley, Reading, USA.

[3] P.O. Bengtsson and J. Bosch. Architecture Level Prediction of Software Maintenance. *Proceedings of the International Conference on Software Engineering '99*. 1999.

[4] F.P. Brooks . *The Mythical Man-Month - Essays on Software Engineering (20th Anniversary Edition)*. Addison Wesley, Reading, USA. 1995.

[5] J.C. Dueñas, W.L. de Olivieira and J.A. de la Puenta. A Software Architecture Evaluation Model. *Proceedings of the Second International ESPRIT Workshop*. Springer Verlag. 1998.

[6] V. Gruhn and U. Wellen. Integration of Heterogenous Software Architectures - An Experience Report. In: P. Donohoe (ed.). *Software architecture: Proceedings of the First Working IFIP Conference on Software Architecture*. Kluwer Academic Publishers, Dordrecht, The Netherlands. 1999.

[7] R. Kazman, G. Abowd, L. Bass and P. Clements. Scenario-Based Analysis of Software Architecture. *IEEE Software*, 13 (6). 1996. pp 47-56.

[8] N.H. Lassing, D.B.B. Rijsenbrij and J.C. van Vliet. Flexibility of the ComBAD architecture. In: P. Donohoe (ed.). *Software architecture: Proceedings of the First Working IFIP Conference on Software Architecture*. Kluwer Academic Publishers, Dordrecht, The Netherlands. 1999.